Tensorflow 可视化 TensorBoard 尝试

数据、模型可视化是TensorFlow的一项重要的功能，安装后自带的TensorBoard是一个很强大的工具，但目前的教程大多都停留在TensorFlow 1.0 版本之前，一些函数已经改名无法使用，因此写一篇比较新的使用说明。

主要区别

如果之前使用过TensorBoard，其实只是换一下函数名就可以了。在Github上新版本说明文档中，已经有了对这一方面的说明：

> Replace tf.scalar_summary, tf.histogram_summary, tf.audio_summary, tf.image_summary with tf.summary.scalar, tf.summary.histogram, tf.summary.audio, tf.summary.image, respectively. The new summary ops take name rather than tag as their first argument, meaning summary ops now respect TensorFlow name scopes.

也就是说，summary独立出来了，以前 `tf.XXX_summary` 这样的下划线变成了 `tf.summary.XXX` 的格式。

数据可视化

对于标量

如果我们想对标量在训练中可视化，可以使用 `tf.summary.scalar()` ，比如损失loss：

```
loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys-prediction),reduction_indices=[1]))
tf.summary.scalar('loss',loss)
```

得到一个loss的summary。

对于参数

应使用 `tf.summary.histogram()` ，如全链接的权重：

```
tf.summary.histogram("/weights",Weights)
```

merge并运行

就像变量需要初始化一样，summary也需要merge：

```
merged = tf.summary.merge_all()
```

之后定义一个输出器记录下在运行中的数据：

```
writer = tf.summary.FileWriter("output/",sess.graph)
```

最后记得在训练过程中执行这两个模块：

TensorBoard 运行

安装TensorFlow时已经自带TensorBoard，如果直接在命令行中输入 `tensorboard` 而没有对应指令，可以从安装目录下执行：

```
python ~/.local/lib/python2.7/site-packages/tensorflow/tensorboard/tensorboard.py --logdir=output/
```

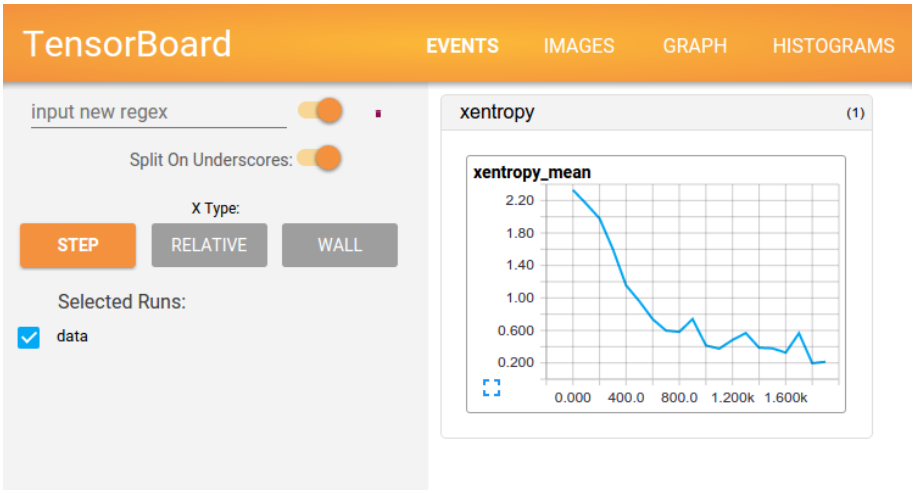
运行成功后，会显示：

```
(You can navigate to http://XXX.XXX.XXX.XXX:6006)
```

然后在浏览器中输入这个地址即可。

为了更方便 TensorFlow 程序的理解、调试与优化，我们发布了一套叫做 TensorBoard 的可视化工具。你可以用 TensorBoard 来展现你的 TensorFlow 图像，绘制图像生成的定量指标图以及附加数据。

当 TensorBoard 设置完成后，它应该是这样子的：



数据序列化

TensorBoard 通过读取 TensorFlow 的事件文件来运行。TensorFlow 的事件文件包括了你会在 TensorFlow 运行中涉及到的主要数据。下面是 TensorBoard 中汇总数据（Summary data）的大体生命周期。

首先，创建你想汇总数据的 TensorFlow 图，然后再选择你想在哪个节点进行汇总(summary)操作。

比如，假设你正在训练一个卷积神经网络，用于识别 MNISt 标签。你可能希望记录学习速度(learning rate)的如何变化，以及目标函数如何变化。通过向节点附加scalar_summary操作来分别输出学习速度和期望误差。然后你可以给每个 scalary_summary 分配一个有意义的 `标签`，比如 `'learning rate'` 和 `'loss function'`。

或者你还希望显示一个特殊层中激活的分布，或者梯度权重的分布。可以通过分别附加 histogram_summary 运算来收集权重变量和梯度输出。

所有可用的 summary 操作详细信息，可以查看summary_operation文档。

在TensorFlow中，所有的操作只有当你执行，或者另一个操作依赖于它的输出时才会运行。我们刚才创建的这些节点（summary nodes）都围绕着你的图像：没有任何操作依赖于它们的结果。因此，为了生成汇总信息，我们需要运行所有这些节点。这样的手动工作是很乏味的，因此可以使用tf.merge_all_summaries来将他们合并为一个操作。

然后你可以执行合并命令，它会依据特点步骤将所有数据生成一个序列化的 `Summary` protobuf对象。最后，为了将汇总数据写入磁盘，需要将汇总的protobuf对象传递给tf.train.Summarywriter。

现在已经修改了你的图，也有了 `SummaryWriter` ，现在就可以运行你的神经网络了！如果你愿意的话，你可以每一步执行一次合并汇总，这样你会得到一大堆训练数据。这很有可能超过了你想要的数据量。你也可以每一百步执行一次合并汇总，或者如下面代码里示范的这样。

```
merged_summary_op = tf.merge_all_summaries()
summary_writer = tf.train.SummaryWriter('/tmp/mnist_logs', sess.graph)
total_step = 0
while training:
  total_step += 1
  session.run(training_op)
  if total_step % 100 == 0:
    summary_str = session.run(merged_summary_op)
    summary_writer.add_summary(summary_str, total_step)
```

现在已经准备好用 TensorBoard 来可视化这些数据了。

启动TensorBoard

输入下面的指令来启动TensorBoard

```
python tensorflow/tensorboard/tensorboard.py --logdir=path/to/log-directory
```

这里的参数 `logdir` 指向 `SummaryWriter` 序列化数据的存储路径。如果 `logdir` 目录的子目录中包含另一次运行时的数据，那么 TensorBoard 会展示所有运行的数据。一旦 TensorBoard 开始运行，你可以通过在浏览器中输入 `localhost:6006` 来查看 TensorBoard。

如果你已经通过pip安装了 TensorBoard，你可以通过执行更为简单地命令来访问 TensorBoard

```
tensorboard --logdir=/path/to/log-directory
```

进入 TensorBoard 的界面时，你会在右上角看到导航选项卡，每一个选项卡将展现一组可视化的序列化数据集 。对于你查看的每一个选项卡，如果 TensorBoard 中没有数据与这个选项卡相关的话，则会显示一条提示信息指示你如何序列化相关数据。

TensorFlow自带的一个强大的可视化工具

功能

这是TensorFlow在MNIST实验数据上得到Tensorboard结果

- Event: 展示训练过程中的统计数据（最值，均值等）变化情况
- Image: 展示训练过程中记录的图像
- Audio: 展示训练过程中记录的音频
- Histogram: 展示训练过程中记录的数据的分布图

原理

- 在运行过程中，记录结构化的数据
- 运行一个本地服务器，监听6006端口
- 请求时，分析记录的数据，绘制

实现

在构建graph的过程中，记录你想要追踪的Tensor

```
with tf.name_scope('output_act'):
    hidden = tf.nn.relu6(tf.matmul(reshape, output_weights[0]) + output_biases)
    tf.histogram_summary('output_act', hidden)
```

其中，

- histogram_summary用于生成分布图，也可以用scalar_summary记录存数值
- 使用scalar_summary的时候，tag和tensor的shape要一致
- name_scope可以不写，但是当你需要在Graph中体现tensor之间的包含关系时，就要写了，像下面这样：

```
with tf.name_scope('input_cnn_filter'):
    with tf.name_scope('input_weight'):
        input_weights = tf.Variable(tf.truncated_normal(
0.1            [patch_size, patch_size, num_channels, depth], stddev=), name='input_weight')
```

```
        variable_summaries(input_weights, 'input_cnn_filter/input weight')
    with tf.name_scope('input_biases'):
        input_biases = tf.Variable(tf.zeros([depth]), name='input_biases')
        variable_summaries(input_weights, 'input_cnn_filter/input_biases')
```

- 在Graph中会体现为一个input_cnn_filter，可以点开，里面有weight和biases
- 用summary系列函数记录后，Tensorboard会根据graph中的依赖关系在Graph标签中展示对应的图结构
- 官网封装了一个函数，可以调用来记录很多跟某个Tensor相关的数据：

```
def variable_summaries(var, name):
    """Attach a lot of summaries to a Tensor."""
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.scalar_summary('mean/' + name, mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_sum(tf.square(var - mean)))
        tf.scalar_summary('sttdev/' + name, stddev)
        tf.scalar_summary('max/' + name, tf.reduce_max(var))
        tf.scalar_summary('min/' + name, tf.reduce_min(var))
        tf.histogram_summary(name, var)
```

- 只有这样记录国max和min的Tensor才会出现在Event里面
- Graph的最后要写一句这个，给session回调

```
merged = tf.merge_all_summaries()
```

Session 中调用

- 构造两个writer，分别在train和valid的时候写数据：

```
train_writer = tf.train.SummaryWriter(summary_dir + '/train',
                                      session.graph)
valid_writer = tf.train.SummaryWriter(summary_dir + '/valid')
```

- 这里的summary_dir存放了运行过程中记录的数据，等下启动服务器要用到
- 构造run_option和run_meta，在每个step运行session时进行设置：

```
summary, _, l, predictions =
    session.run([merged, optimizer, loss, train_prediction], options=run_options, feed_dict=feed_dict)
```

- 注意要把merged拿回来，并且设置options
- 在每次训练时，记一次：

```
train_writer.add_summary(summary, step)
```

- 在每次验证时，记一次：

```
valid_writer.add_summary(summary, step)
```

- 达到一定训练次数后，记一次meta做一下标记

```
train_writer.add_run_metadata(run_metadata, 'step%03d' % step)
```

查看可视化结果

- 启动TensorBoard服务器：

```
python安装路径/python TensorFlow安装路径/tensorflow/tensorboard/tensorboard.py --logdir=path/to/log-directory
```

注意这个**Python**必须是安装了TensorFlow的python，tensorboard.py必须制定路径才能被python找到，logdir必须是前面创建两个writer时使用的路径

比如我的是：

```
/home/cwh/anaconda2/envs/tensorflow/bin/python /home/cwh/anaconda2/envs/tensorflow/lib/python2.7/site-packages/tensorflow/tensorboard/tensorboard.py --logdir=~/coding/python/GDLnotes/src/convnet/summary
```

使用python

- 然后在浏览器输入 http://127.0.0.1:6006 就可以访问到tensorboard的结果

参考资料

- mnist_with_summaries.py

> 觉得我的文章对您有帮助的话，不妨点个star？

mnist_with_summaries.py的源码如下：

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the 'License');
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an 'AS IS' BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
"""A simple MNIST classifier which displays summaries in TensorBoard.
 This is an unimpressive MNIST model, but it is a good example of using
tf.name_scope to make a graph legible in the TensorBoard graph explorer, and of
naming summary tags so that they are grouped meaningfully in TensorBoard.
It demonstrates the functionality of every TensorBoard dashboard.
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data

flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_boolean('fake_data', False, 'If true, uses fake data '
                     'for unit testing.')
flags.DEFINE_integer('max_steps', 1000, 'Number of steps to run trainer.')
flags.DEFINE_float('learning_rate', 0.001, 'Initial learning rate.')
flags.DEFINE_float('dropout', 0.9, 'Keep probability for training dropout.')
flags.DEFINE_string('data_dir', '/tmp/data', 'Directory for storing data')
flags.DEFINE_string('summaries_dir', '/tmp/mnist_logs', 'Summaries directory')


def train():
  # Import data
  mnist = input_data.read_data_sets(FLAGS.data_dir,
                                    one_hot=True,
                                    fake_data=FLAGS.fake_data)

  sess = tf.InteractiveSession()

  # Create a multilayer model.

  # Input placehoolders
  with tf.name_scope('input'):
    x = tf.placeholder(tf.float32, [None, 784], name='x-input')
    y_ = tf.placeholder(tf.float32, [None, 10], name='y-input')

  with tf.name_scope('input_reshape'):
    image_shaped_input = tf.reshape(x, [-1, 28, 28, 1])
    tf.image_summary('input', image_shaped_input, 10)

  # We can't initialize these variables to 0 - the network will get stuck.
  def weight_variable(shape):
    """Create a weight variable with appropriate initialization."""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
```

```
def bias_variable(shape):
  """Create a bias variable with appropriate initiali
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

def variable_summaries(var, name):
  """Attach a lot of summaries to a Tensor."""
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.scalar_summary('mean/' + name, mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_sum(tf.square(var - mean)))
    tf.scalar_summary('sttdev/' + name, stddev)
    tf.scalar_summary('max/' + name, tf.reduce_max(var))
    tf.scalar_summary('min/' + name, tf.reduce_min(var))
    tf.histogram_summary(name, var)

def nn_layer(input_tensor, input_dim, output_dim, layer_name, act=tf.nn.relu):
  """Reusable code for making a simple neural net layer.
  It does a matrix multiply, bias add, and then uses relu to nonlinearize.
  It also sets up name scoping so that the resultant graph is easy to read,
  and adds a number of summary ops.
  """
  # Adding a name scope ensures logical grouping of the layers in the graph.
  with tf.name_scope(layer_name):
    # This Variable will hold the state of the weights for the layer
    with tf.name_scope('weights'):
      weights = weight_variable([input_dim, output_dim])
      variable_summaries(weights, layer_name + '/weights')
    with tf.name_scope('biases'):
      biases = bias_variable([output_dim])
      variable_summaries(biases, layer_name + '/biases')
    with tf.name_scope('Wx_plus_b'):
      preactivate = tf.matmul(input_tensor, weights) + biases
      tf.histogram_summary(layer_name + '/pre_activations', preactivate)
    activations = act(preactivate, 'activation')
    tf.histogram_summary(layer_name + '/activations', activations)
    return activations

hidden1 = nn_layer(x, 784, 500, 'layer1')

with tf.name_scope('dropout'):
  keep_prob = tf.placeholder(tf.float32)
  tf.scalar_summary('dropout_keep_probability', keep_prob)
  dropped = tf.nn.dropout(hidden1, keep_prob)

y = nn_layer(dropped, 500, 10, 'layer2', act=tf.nn.softmax)

with tf.name_scope('cross_entropy'):
  diff = y_ * tf.log(y)
  with tf.name_scope('total'):
    cross_entropy = -tf.reduce_mean(diff)
  tf.scalar_summary('cross entropy', cross_entropy)

with tf.name_scope('train'):
  train_step = tf.train.AdamOptimizer(FLAGS.learning_rate).minimize(
      cross_entropy)

with tf.name_scope('accuracy'):
  with tf.name_scope('correct_prediction'):
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  with tf.name_scope('accuracy'):
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  tf.scalar_summary('accuracy', accuracy)

# Merge all the summaries and write them out to /tmp/mnist_logs (by default)
merged = tf.merge_all_summaries()
train_writer = tf.train.SummaryWriter(FLAGS.summaries_dir + '/train',
                                      sess.graph)
test_writer = tf.train.SummaryWriter(FLAGS.summaries_dir + '/test')
tf.initialize_all_variables().run()

# Train the model, and also write summaries.
# Every 10th step, measure test-set accuracy, and write test summaries
# All other steps, run train_step on training data, & add training summaries

def feed_dict(train):
  """Make a TensorFlow feed_dict: maps data onto Tensor placeholders."""
  if train or FLAGS.fake_data:
    xs, ys = mnist.train.next_batch(100, fake_data=FLAGS.fake_data)
    k = FLAGS.dropout
```

```
    else:
      xs, ys = mnist.test.images, mnist.test.labels
      k = 1.0
    return {x: xs, y_: ys, keep_prob: k}

  for i in range(FLAGS.max_steps):
    if i % 10 == 0:  # Record summaries and test-set accuracy
      summary, acc = sess.run([merged, accuracy], feed_dict=feed_dict(False))
      test_writer.add_summary(summary, i)
      print('Accuracy at step %s: %s' % (i, acc))
    else:  # Record train set summaries, and train
      if i % 100 == 99:  # Record execution stats
        run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
        run_metadata = tf.RunMetadata()
        summary, _ = sess.run([merged, train_step],
                              feed_dict=feed_dict(True),
                              options=run_options,
                              run_metadata=run_metadata)
        train_writer.add_run_metadata(run_metadata, 'step%d' % i)
        train_writer.add_summary(summary, i)
        print('Adding run metadata for', i)
      else:  # Record a summary
        summary, _ = sess.run([merged, train_step], feed_dict=feed_dict(True))
        train_writer.add_summary(summary, i)


def main(_):
  if tf.gfile.Exists(FLAGS.summaries_dir):
    tf.gfile.DeleteRecursively(FLAGS.summaries_dir)
  tf.gfile.MakeDirs(FLAGS.summaries_dir)
  train()


if __name__ == '__main__':
  tf.app.run()
```

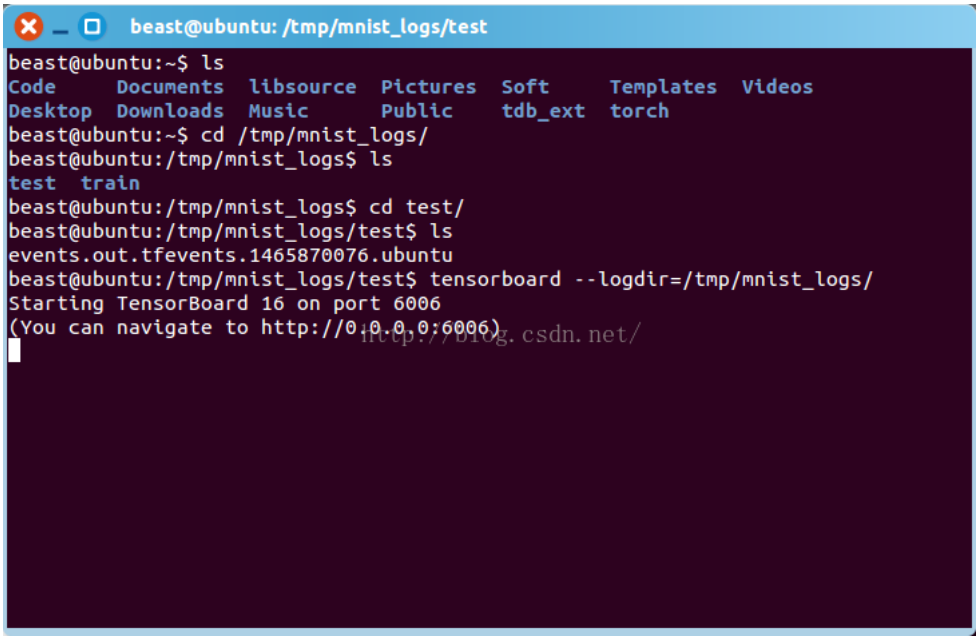
其中

```
flags.DEFINE_string('summaries_dir', '/tmp/mnist_logs', 'Summaries directory')
```

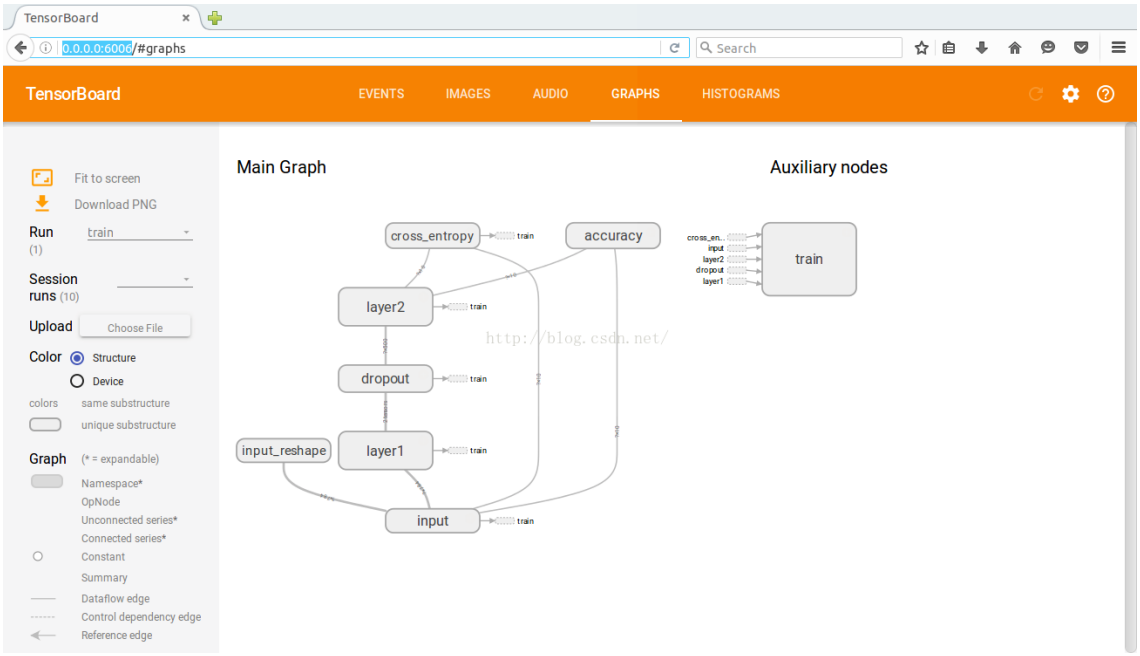
标识了事件文件的输出路径。该例中，输出路径为/tmp/mnist_logs

打开TensorBoard服务

```
tensorboard --logdir=/tmp/mnist_logs/
```



在浏览器中进行浏览http://0.0.0.0:6006，在这个可视化界面中，可以查看tensorflow图和各种中间输出等。



TensorBoard的不过是个调试工具，看起来很酷炫有没有，但怎么充分利用，我想还是要对tensorflow充分了解。下面要转向对tensorflow的学习中了。